České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Interaktivní nástroj pro kreslení schémat logických obvodů

Rastislav Pastor

Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika strukturovaný bakalářský

Obor: Informatika a výpočetní technika

červen 2006

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve zmyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne

.....

Podpis

.....

Anotácia

V tejto práci sa venujem teoretickému popisu tvorby EDA (Electronic Design Automation) aplikácie pre kreslenie schém logických obvodov. Rozoberám jednotlivé stavebné prvky tvoriace aplikáciu a uvádzam možné spôsoby ich realizácie spolu s vnodnosťou takejto realizácie. Stručne popisujem implementáciu mnou vytvorenej verzie aplikácie a hlavných tried v nej využitých.

Abstract

This thesis presents theoretical description of creating EDA (Electronic Design Automation) application for creating logicar circuit schemes. Structural elements creating such application are described here along with possible methods of implementation and applicability of such implementation. I briefly described the very implementation of application and main classes used.

Obsah

Zozam obrázkov

1. Úvod

1.1. Zadanie projektu

Zadaním tohto projektu je:

- Vytvořte nástroj pro interaktivní kreslení schémat logických obvodů. V C++
- Vstup: schéma nakreslené uživatelem
- Výstup: PS, netlist
- GUI bude striktně oddělené od jádra
- Interaktivní nástroj - bude možnost zpětné editace schématu
- Nástroj nebude sám kreslit schémata z netlistu
- Možnost napojení na nástroj kreslící schéma z netlistu
- Podpora hierarchie ve schématech
- Rozšíření - možnost rozpoznání netlistu z PS

1.2. Základná myšlienka projektu

Základnou myšlienkou tohto projektu je vytvorenie nástroja pre interaktívne kreslenie schém, teda grafickej reprezentácie logických obvodov. Ide o vytvorenie aplikácie, ktorá umožní uživateľovi kresliť schémy logických obvodov skladajúcich sa zo základnej množiny elementov (logických hradiel) navzájom prepojených spojmi a súborormi spojov. Nástroj, ktorý umožňuje vytvorenie grafickej reprezentácie schémy, je dôležitou pomôckou pri jej návrhu. Ľudský mozog totiž dokáže spracovať oveľa viac informácií vo vizuálnej podobe, než čítaním textovej reprezentácie schémy. Vstupom tejto aplikácie je teda schéma nakreslená uživateľom. Pri jej kreslení má uživateľ možnosť využiť preddefinovanú sadu zobrazení základných logických hradiel a blokov, prípadne vytvoriť si vlastnú sadu zobrazení týchto elementov a pracovať s nimi. Nástroj podporuje hierarchiu, je teda možné upravovať nielen obsah schémy na úrovni samotných logických hradiel, ale aj obsah funkčných blokov obsiahnutých v schéme (skladajúcich sa z logických hradiel, prípadne ďalších funkčných blokov). Vytvorenú schému môže uživateľ uložiť do súboru a spätne načítať aplikáciou, teda spätne upravovať.

Pri úvodnej úvahe nad realizáciou tohto projektu som prišiel na mnoho komplikácií, ktoré bude potrebné vyriešiť. Tieto komplikácie sa nevyskytujú len pri tvorbe jednotlivých zložiek aplikácie, ako je napríklad interakcia uživateľa s aplikáciou, zobrazenie grafických prvkov aplikáciou, ich správne uloženie a manipulácia s nimi, alebo obsluha GUI prvkov aplikácie. Dôležitým a neľahkým aspektom je aj prepojenie jednotlivých týchto samostatných častí v jednen funkčný celok.

V tejto práci sú uvedené otázky a ich možné riešenia, súvisiace s návrhom takejto aplikácie, popis mnou vybraných riešení pri návrhu a implementácii projektu a krátky popis implementácie.

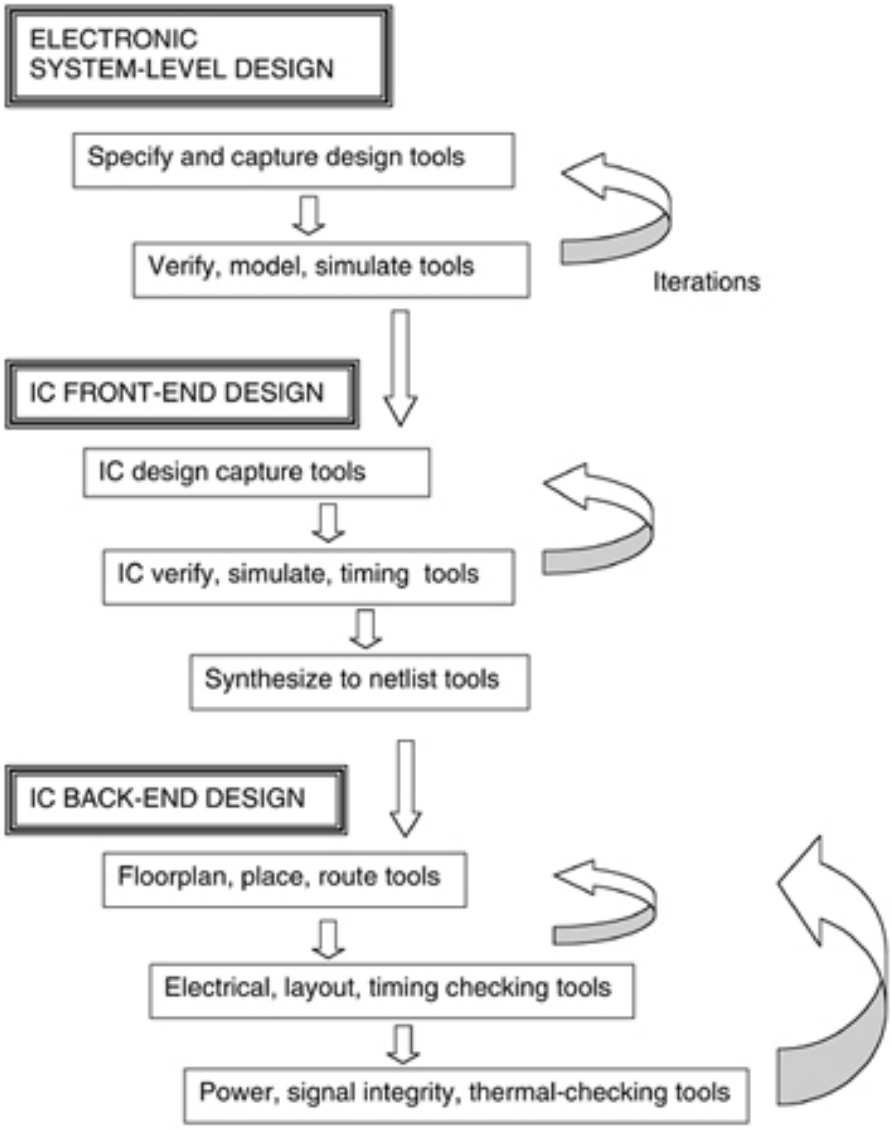
2. Popis problému

Táto časť textu špecifikuje zaradenie vytváraneho projektu do kategórie, pojednáva o požiadavkoch projektu, otázkach súvisiacich s kladenými požiadavkami, návrhom projektu, o problémoch, ktoré sa vyskytujú pri tvorbe takéhoto typu aplikácie. Taktiež určuje hranice vymedzujúce oblasť zamerania sa v tomto projekte.

2.1. Všeobecný náhľad

Požiadavky tejto aplikácie naznačujú, že ide o tzv. EDA (Electronic Design Automation) nástroj. Táto skratka zastrešuje kategóriu nástrojov pre návrh a tvorbu elektronických systémov. Elektronické systémy predstavujú široké spektrum produktov, ako napríklad mobilné telefóny, digitálne fotoaparáty, komunikačné zariadenia. Tieto produkty v sebe obsahujú tlačené spoje resp. integrované obvody, pri návrhu ktorých sa využívajú práve EDA nástroje. Medzi hlavné zložky EDA nástrojov patrí skupina ESL (Electronic System Level) pre tvorbu modelov vytváraného systému, pomáhajúcich pri zisťovaní vhodnosti použitých materiálov, správneho rozloženia funkcií medzi hardware a software, prípadne správnosti funkcionality systému. Táto počiatočná fáza zabezpečuje zahrnutie všetkých zadaných požiadavkov na systém do procesu jeho tvorby. Druhou skupinou sú nástroje pre tvorbu front-end návrhu, zahŕňajúce nástroje pre návrh schém (grafický, textový), verifikáciu a syntézu. V tejto fáze je zachytený návrh usporiadania elementov v schéme, ktorého funkčnost je následne testovaná verifikačnými nástrojmi prevádzajúcimi simuláciu návrhnu, kontrolujúc správnosť správania sa a vykonávania požadovaných funkcií obvodom. V poslednej časti front-end návrhu, syntéze, je grafické zobrazenie schémy rozložené na základné komponenty - symboly. Tieto symboly nezodpovedajú vzhľadu fyzického zariadenia, ktoré reprezentujú, preto je k ďalšej fáze potrebné symboly previesť na textový popis, špecifický pre každé konkrétne zariadenie, určujúce napríklad výšku, šírku symbolu s ohľadom na skutočné fyzické parametre. Vzniká tak výstup front-end návrhu, netlist, ako deliaca čiara medzi front-end a back-end návrhom. Treťou skupinou EDA nástrojov, back-end návrhové nástroje, sú používané pre fyzické rozmiestnenie komponent v obvode (floorplan). Pomáhajú tiež pri návrhu smerovania prepojených ciest a výpočte časových oneskorení v obvode. Každá z týchto fáz je teda tvorená

návrhom špecifickej časti systému s následnou kontrolou funkčnosti návrhu. V prípade zistenia chyby v návrhu je možné v danej fáze návrh preskúmať, pozmeniť a znova testovať jeho funkčnosť. Je dokonca možné vrátiť sa k návrhu ľubovoľnej predošlej fázy v prípade výskytu chyby.



Orázok 1. – Postup využitia EDA nástrojov pri tvorbe systému.

Tento projekt, ako naznačuje jeho základná myšlienka, patrí do druhej skupiny nástrojov návru systému, teda front-end.

2.2. Požiadavky projektu

Cieľom tohto projektu je vytvoriť aplikáciu umožňujúcu kresliť schémy logických obvodov s možnosťou spätnej editácie schémy. Tento základný požiadavok v sebe zahŕňa potrebu schopnosti uloženia vnútorných informácií o uživateľom nakreslenej schéme, ako aj schopnosť jej spätného načítania aplikáciou. Je preto potrebné pre tento projekt navrhnúť štruktúru, spôsob a miesto ukladania týchto vnútorných informácií. K tomuto účelu existuje rada súborových formátov, ktoré sme s p. Ing. Fišerom zúžili na textové formáty, z dôvodu jednoduchosti ich tvorby. Ďalej pojem kreslenia schémy, teda množiny objektov, ktorých grafická reprezentácia je komplexnejšia, skladajúca sa z viacerých čiar, obdĺžnikov, prípadne textu, znamená, že je potrebné toto grafické znázornenie objektov definovať textovým popisom a teda vzniká požiadavok na rozpoznanie definovaných grafických objektov aplikáciou a prevod do grafickej reprezentácie. Aplikácia má obsahovať podporu hierarchie v schémach. Tá je zabezpečená združovaním schémy do funkčných celkov, blokov, s možnosťou znovupoužitia tohto bloku pri ďalšom kreslení. Výstup aplikácie, či už PostScript, alebo netlist je textovým výstupom, z čoho vyplýva nutnosť jeho generovania. Tieto požiadavky sú predstavované mnohými malými podproblémami, ktoré je potrebné riešiť samostatne.

2.3. Vymedzenie cieľov projektu

Aplikácia bude schopná kresliť grafické objekty definované v textovej podobe. Definícia takéhoto objektu sa bude skladať zo základných grafických elementov, teda čiary, elipsy, oblúku, bézierových kriviek a textu. Základné logické hradlá budú preddefinované v aplikácii, pripravené pre použitie uživateľom. K umožneniu kreslenia schémy bude potrebné otvoriť exsitujúci projekt, prípadne vytvoriť nový projekt, do ktorého bude možné pridať túto schému. Vytvorenie funkčného bloku bude možné z existujúcej schémy, ktorá obsahuje aspoň jednu vstupnú, alebo výstupnú značku. Takto vytvorený funkčný blok bude možné používať vrámci projektu pri ďalšom kreslení schém. Počas práce s výslednou aplikáciou bude možné zistiť prepojenia každého objektu v schéme. Aplikácia bude podporovať prevod obsahu schémy a bloku do vektorového formátu, konkrétne postscript a EMF (Microsoft Enhanced Metafile). Pri ukončení aplikácie bude možnosť výslednú prácu uživateľa uložiť do súboru, pre prípadné ďalšie úpravy tejto práce. Takto vytvorené

súbory bude možné použiť aj ako netlist, pretože vyhovujú požiadavkam kladeným na obsah netlistu. Časť verifikácie patriaca do fázy front-end návrhu v aplikácii podporovaná nebude.

2.4. Formulácia problému

Všeobecné požiadavky na aplikáciu je možné zhrnúť do oblastí zamerania:

- základná kostra aplikácie
- GUI prvky, interakcia uživateľa s GUI, obsluha GUI prvkov
- formát vstupu/výstupu vnútorných informácií aplikácie
- reprezentácia, špecifikácia, uloženie jednotlivých grafických objektov
- spôsob riešenia hierarchie schémy
- oddelenie zdrojového kódu pre uchovávanie dát od kódu pre ich zobrazenie

3. Analýza a návrh riešenia

Táto časť textu pojednáva o možných spôsoboch riešenia problémov súvisiacich s vývojom aplikácie, zároveň naznačuje vhodnosť použitia týchto riešení.

3.1. Objekty v schéme

3.1.1. Uloženie objektov

Dôležitým požiadavkom na aplikáciu je schpnosť obsiahnut teoreticky nekonečný počet kreslených objektov. Uživateľ nesmie byť pri vytváraní schémy obmedzovaný pevnými hranicami aplikácie (úložným priestorom pre kreslené objekty s pevne definovanou veľkosťou). Exituje predpoklad násobného umiestňovania objektu do schémy, ako aj násobného mazania objektu. Je preto potrebné dosiahnuť flexibilitu aplikácie pri práci s objektami. Všeobecne je nevýhodné v aplikácii používať pole pevnej veľkosti, dátovej štruktúry použiteľnej ako priestor pre ukladanie objektov, z hľadiska časovej náročnosti dealokácie a následnej alokácie väčšej časti pamäti v prípade zaplnenia poľa. Z dôvodu potreby prístupu náhodnému prvku je nutné vylúčiť použitie dátových štruktúr fronta a zásobník, kde z definície prvého prípadu vyplýva možnosť prístupu len k prvému vloženému prvku a z definície druhého možnosť prístupu len k poslednému vloženému prvku. Ako vhodným kandidátom pre úložný priestor objektov aplikácie sa javí asociatívna (hashovacia) mapa, s prístupom ku samotnému objektu pomocou kľúča tvoreného názvom objektu. Vloženie prvku do mapy sa vykonáva v najlepšom prípade v konštantnom čase. Pre vyhľadanie prvku je očakávaný potrebný čas taktiež konštantný. V oboch prípadoch tento čas závisí na spôsobe implementácie mapy, konkrétne hashovacej funkcie. V nemalom počte implementácií tieto hodnoty rastú na zložitosť O(log(n)). Asymptotická zložistosť vymazania kľúča z mapy je O(log(n)), kde n predstavuje počet prvkov mapy. Ďalšou možnosťou je využitie dátovej štruktúry spojového zoznamu. Pri kreslení objektov nieje vyžadovaný žiadny spôsob usporiadania pri vkladaní do spojového zoznamu, preto je táto operácia vykonávaná s asymptotickou zložitosťou O(1). Pri vymazaní prvku zo spojového zoznamu dochádza len k prepisovaniu ukazateľov, preto je aj táto operácia vykonávaná v konštantnom čase. Vymazaniu prvku však predchádza jeho vyhľadanie, ktoré je v tomto prípade predstavované asymptotickou zložistosťou

najviac O(n), kde n je počet prvkov v spojovom zozname. Toto vyhľadanie prvku znamená pri spojovom zozname náhodný prístup k prvku, ktorý je možné v najhoršom prípade prevádzať sekvenčne. Pri implementácii je možné použiť obe z týchto spomínaných možností, rozdiel bude pri práci s nimi tvoriť vyhľadanie a vymazanie prvku. Aj cez výhodu v prospech asociatívnej mapy som sa rozhodol v implementácii pre uloženie objektov využívať spojový zoznam.

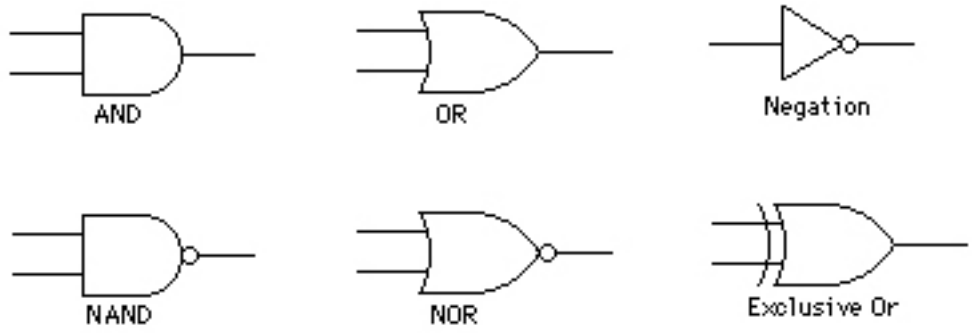
Objekty, ktoré môže schéma obsahovať, je možné rozdeliť do skupín. Patrí sem skupina logických hradiel, teda základných symbolov, skupina blokov, skupina spojov a skupina značiek. Toto rozdelenie vychádza z účelu samotného objektu a z tohto účelu vyplývajúceho spôsobu jeho obsluhy. Každý z objektov musí mať definovaný svoj prototyp určujúci jeho základné vlastnosti.

3.1.2. Grafická reprezentácia objektov

Z hľadiska funkcionality aplikácie, prípadne schémy nieje spôsob vykreslenia (tvar) ľubovoľného z objektov podstatný. Zobrazované objekty vyžadujú existenciu svojho grafického prototypu, teda textového popisu zobrazenia objektu, ktorý je súčasťou samotného prototypu objektu. Výnimkou v tomto pravidle sú objekty typu spoj. Je teda možné jeden objekt reprezentovať viacerými grafickými prototypmi. Uvedené skupiny objektov je možné z hľadiska ich grafickej reprezenácie združiť na základe rozdielnej veľkosti potreby a taktiež možnosti vyjadriť grafický prototyp ich zobrazenia explicitne. Konkrétne sa jedná o sadu logických hradiel a blokov, kde táto potreba explicitného definovania zobrazenia prototypu existuje. Možnosť zobrazenia je taktiež kladná, nieje totiž problémom hradlá a bloky popísať základnými grafickými elementami. Táto sada je teda vytváraná objektami splňajúceho kritérium kladného vyhovenia potrebe a možnosti grafického vyjadrenia. Druhou sadou sú značky a spoje. V prípade značiek možnosť vyjadrenia objektu určite existuje, avšak odpoveď na otázku potreby explicitnej definície zobrazenia je nejednoznačná. Zaradenie do druhej sady vzniklo na základe môjho osobného názoru malej, až žiadnej veľkosti tejto potreby. Pre spoje platí jednoznačná nemožnosť definovať grafický prototyp objektu, čo vyplýva zo samozrejmej rôznorodosti tvaru spoja.

3.1.2.1. Grafická reprezentácia logických hradiel a blokov

Každý objekt zo skupiny základných logických hradiel a blokov musí mať explicitne definovaný aspoň jeden grafický prototyp. Špecifikácia grafického prototypu, teda popis grafickej reprezenácie, by mal byť rozdelený do základných grafických elementov, ako sú čiary, elipsy, bézierové krivky a text k dosiahnutiu maximálnej univerzálnosti pri vytváraní tejto reprezentácie.



Orázok 2. – Základné logické hradlá a ich možné grafické znázornenie

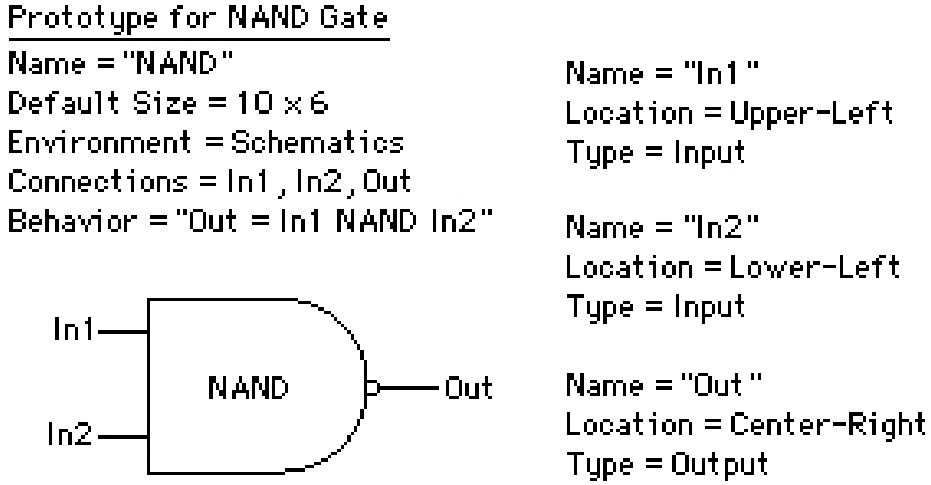
3.1.2.2. Grafická reprezentácia značiek a spojov

Značky a spoje patria do sady objektov, ktoré nepotrebujú pre svoju existenciu a teda zobrazenie v schéme vlastný grafický prototyp. Pre značky je to z dôvodu malej potreby viacerých grafických reprezentácií. Je však potrebná akási náhrada grafického prototypu, ktorá musí byť špecifikovaná aplikáciou, vykonávajúca podobnú funkciu ako grafický prototyp definovaný textovým súborom. Pre spoje, vznikajúce pri ich kreslení postupným pridávaním bodov, ktorých minimálny počet je dva (vytvárajúcich úsečku), grafický prototyp nieje možné vytvoriť. Dôvodom je ľubovoľný počet bodov (a teda úsečiek), ktoré spoj tvoria, a taktiež ľubovoľné umiestnenie každého z bodov.

3.1.3. Prototypy objektov

Vytvorenie prototypov objektov je dôležitou oblasťou pri tvorbe takejto aplikácie. Každý prototyp objektu nesie špecifické informácie, napríklad o počte vstupov, výstupov, type objektu, umiestnení v schéme, hlavne však o jeho grafickej

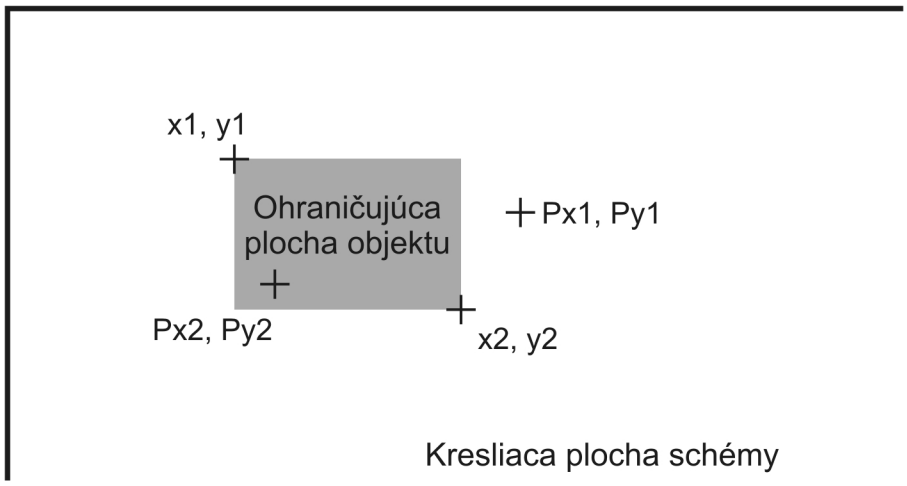
reprezentácii. Tieto informácie je potrebné získavať z textového popisu objektu. Potreba prototypov spočíva vo vysokej časovej neefektivite znovuzískavania týchto informácií zo súboru a následného vytvárania samotného kresleného objektu v prípade neexistencie prototypu. Hoci prototypy nesú informácie o ich grafickej reprezentácii, oni samotné nebudú nikdy aplikáciou zobrazované. Je vhodné uložiť ich v pamäti aplikácie počas doby jej behu a v momente vloženia (kreslenia) objektu do schémy je potrebné konkrétny prototyp duplikovať, čím sa vytvorí objekt, ktorý je už zobrazovaný a s ktorým je možné ďalej pracovať.



Orázok 3. – Príklad prototypu hradla NAND

3.1.4. Označovanie objektov

K základným potrebám pri vytváraní schémy patrí okrem možnosti umiestnenia objektu aj jeho presun, prípadne odstránenie. Tomu predchádza nutnosť jeho označenia, teda výberu jedného objektu z množiny všekých objektov v schéme na základe istého kritéria. Je teda potrebné využívať metódu nazývanú hit-test, ktorej kritériom výberu je správna pozícia kurzoru pri kliknutí tlačidla myši s účelom konkrétny objekt označiť. Existuje veľmi vysoký predpoklad častého využívania metódy hit-test, je preto veľmi podstatné zvoliť správny spôsob jej implementácie k zabezpečeniu časovej efektivity práce s objektami.



Orázok 4. – Princíp jednoduchej metódy hit-test

Základná možnosť návrhu hit-test metódy pozostáva z vytvorenia ohraničujúcej plochy grafickej reprezentácie objektu. Ohraničujúca plocha je predstavovaná obdĺžnikom definovaným suradnicami x1, y1 a x2, y2. Pri kliknutí myšou sú získané súradnice (Px1, Py1 resp. Px2, Py2) ktoré sú testované vysokým počtom podmienok na ich umiestnenie v objekte, výsledok hit-test metódy je teda kladný, alebo mimo objektu, so záporným výsledkom hit-test metódy. Pre potreby aplikácie je vytvorenie ohraničujúcej plochy možné pre všetky kreslené objekty, s výnimkou spojov. Tento špeciálny prípad zahŕňa dve možné situácie, a síce, v prípade, že je spoj tvorený jedinou úsečkou, ktorá je pomerne tenká, nastáva problém zväčšenej obtiažnosti správnosti kliknutia pri snahe označiť spoj. Tento problém je samozrejme možné vyriešiť rozšírením ohraničujúcej plochy spoja. Druhá situácia nastáva v prípade spoja tvoreného viacerými úsečkami s rôznou orientáciou, kde je obtiažné vytvoriť ohraničujúu plochu. Tento prípad je možné riešiť rozdelením spoja na dvojice bodov tvoriacich úsečku vždy v jednom smere a následným testovaním každej z jeho častí. Vysoký počet testovacích podmienok pri kontrole správnosti súradníc bodu vytvoreného kliknutím je jednou z nevýhod takejto metódy, ďalšou je nutnosť vytvorenia ohraničujúcej plochy všetkých objektov, ktoré budú testované, v prípade spojov prípadné rozdelenie na časti. Spolu táto úloha predstavuje asymptotickú zložitosť minimálne O(n) vyplývajúcu zo sekvenčného prechodu objektov v schéme. Všetky spôsoby metódy hit-test založené na sekvenčnom prechode už umiestnených

objektov predstavujú spojenie zložitosti prechodu všetkých objektov a samotného vyhovenia základnému kritériu umiestnenia bodu testovania v grafickej reprezentácii objektu. K dosiahnutiu lepšieho výsledku metódy hit-test je preto potrebné vytvoriť dobré prostredie v ktorom bude pracovať. Dátové štruktúry použiteľné pre uloženie objektov v schéme slúžia len ako úložný priestor, nijakým spôsobom pri vložení objektu nezohľadňujú ich umiestnenie. Riešením vytvorenia prostredia pre metódu hit-test môže teda byť registrácia objektov na základe ich skutočnej polohy v schéme. Možnosťou takejto registrácie je napríklad indikátor výskytu niektorého z objektov schémy na daných súradniciach. Takéto prostredie vyžadujúce správne nastavenie príznaku pri umiestnení objektu ako aj po presune objektu by celkovo urýchlilo hit-test metódu len v prípade, že súradnice nevyhovujú základnému kritériu hit-testu, v opačnom prípade by však nebolo možné zistiť ktorý z objektov bol vybraný. Preto vylepšením tejto metódy môže byť registrácia odkazov na samotné objekty, pre dané súradnice. V prípade spojov je takáto registrácia taktiež možná, netvoria teda výnimku, pretože samotný objekt typu spoj je registrovaný na súradniciach, ktoré zaberá.

3.2. Vstupy a výstupy aplikácie

Po nakreslení samotnej schémy, prípadne jej časti, je potrebné uživateľovi umožňiť vrátiť sa k jej úprave. Pre tento cieľ je potrebné informácie o vytvorenej schéme uložiť do súboru. Pri výbere súborového formátu je vhodné zamerať sa najlepšie na formáty schopné pojať grafické informácie, keďže ide o kreslenie schémy. Rastrový výstup do samozrejme do úvahy neprichádza, kôli potrebe oddelenia jednotlivých objektov schémy, spolu s informáciami o objekte samotnom, od objektov ostatných. Je teda potrebné použiť vektorový formát. Tvorba známych vektorových grafických formátov, ako sú napríklad AI (Adobe Illustrator) alebo DXF (AutoCAD), je pomerne dosť komplexná, rovnako tak ako ich načítanie. Vyžadujú striktný formát vytváraných súborov a ich špecifikácia je rozsiahla. Vytváraný projekt nevyžaduje veľké množstvo funkcií podporovaných týmito formátmi, je preto neefektívne zaoberať sa nimi. V poslednej dobe sa veľmi využívaným metajazykom stáva XML (Extensible Markup Language) ako podtrieda metajazyka SGML (Standard Generalized Markup Language). Tento jazyk je možné vďaka jeho univerzálnosti využívať nielen pre uloženie dát zobrazovaných jazykom HTML, ale taktiež pre jednoduchú výmenu

informácií, alebo ukladanie dát do databázy. Jeho štruktúra podporuje hierarchické ukladanie dát, je prehľadná pri čítaní očami, jednoduchá pre vytvorenie, okrem samotných dát vyžaduje len minimum sprievodných dát. Informácie o nakreslenej schéme, teda dáta aplikácie, skladajúce sa zo zoznamu objektov spolu s ich vlastnosťami je preto veľmi vhodné a jednoduché špecifikovať jazykom XML pri tvorbe výstupu aplikácie. Keďže ide o textový formát výstupu, je potrebné pri jeho načítaní využiť parser ako program analyzujúci tento vstup, spolu s využitím syntaktického analyzátora rozpoznávajúceho symboly gramatiky jazyka XML. Táto úloha nieje náplňou projektu, preto je vhodné využiť už existujúci a správne fungujúci parser jazyka XML.

3.3. Uživateľské rozhranie aplikácie

Pri vývoji akéhokoľvek grafického editora je potrebné prihliadať na uživateľa, ktorý bude výsledný produkt využívať, a snažiť sa poskytnúť mu jednoduchý a prehľadný spôsob jeho obsluhy a teda celkovo práce s ním. K tomu vo veľkej miere prispieva logické usporiadanie jednotlivých ovládacích prvkov, ako aj zahrnutie pre každú aplikáciu bežných prkov. Konkrétne sa jedná o prvky ako sú klávesové skratky, toolbary, prehľadné menu a vhodné kontextové menu. V súčasnej dobe tento projekt nevyžaduje poskytovanie veľkého množstva možností uživateľovi pri tvorbe schémy, avšak je potrebné predpokladať rozširenie jeho funkcionality v budúcnosti. Preto je vhodné už teraz udržiavať prehľadnosť v poskytovaných možnostiach a zároveň každú z týchto možností vhodne prezentovať uživateľovi s využitím spomínaných bežných ovládacích prvkov. Ďalej je vhodné z dôvodu prehľadnosti hierarchického návrhu schémy túto hierarchiu zobrazovať uživateľovi počas jej tvorby. Pre takéto zobrazenie je správne využiť stromovú štruktúru zobrazenia obsiahnutých schém, blokov, logických hradiel, značiek ai.

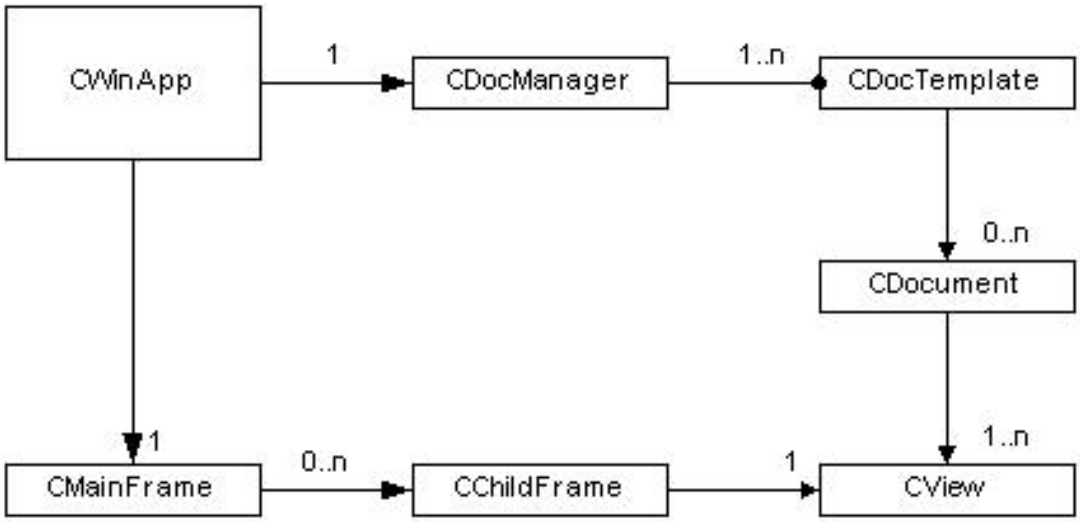
4. Realizácia projektu

4.1. Popis aplikácie

Aplikácia je vytvorená s pomocou knižníc MFC (Microsoft Foundation Classes), založených na Document/View architektúre. Pre možnosť úpravy viacerých schém naraz aplikácia využíva MDI (Multiple Document Interface) architektúru. Prototypy objektov sú definované v XML súboroch, preto na ich načítanie využíva open-source c++ XML parser TinyXml. Objekty kreslené uživateľom sú ukládané do spojového zoznamu.

4.1.1. Štruktúra MFC

Základom MFC aplikácie je jediný objekt typu CWinApp, predstavujúci bežiaci proces. Tento objekt v sebe obsahuje inštanciu triedy CDocManager. Ten je využívaný MFC na obsluhu všetkých CDocTemplate objektov, predstavujúcich typ dokumentu, registrovaných frameworkom. CWinApp objekt taktiež vytvára inštanciu triedy CMainFrame, ktorá predstavuje hlavné okno aplikácie. Pri každom vytvorení, alebo otvorení nového dokumentu v aplikácii je vytvorený objekt typu CDocument, s príslušným typom a odkaz na tento objekt je uložený do zoznamu, pod ten CDocTemplate objekt, ktorému typ vytvoreného dokumentu zodpovedá. Pri otvorení dokumentu CMainFrame vytvára inštanciu triedy CChildFrame, teda MDI (Multiple Document Interface) okno používané na zobrazenie pohľadu (CView) príslušného dokumentu. Zároveň je vytvorený objekt typu CView a ukazateľ na neho je uložený do zoznamu pohľadov objektu CDocument.



Orázok 5. – Štruktúra MFC frameworku

4.1.2. Hierarchická štruktúra tried grafických objektov

Rodičovskou triedou všetkých grafických objektov je trieda `CCommonObject`. Obsahuje prototypy funkcií využívaných jej potomkami. Aplikácia podporuje 5 typov zobrazovaných grafických objektov. Ide o inštancie tried `CGate`, `CBlock`, `CWire`, `CTag` a `CInv` reprezentujúcich jednotlivé grafické elementy (konkrétne ide o logické hradlá, funkčné bloky, spoje, značky a invertor). Toto rozdelenie vzniklo na základe rozdielnych vlastností a správania sa objektov, a teda rôznej obsluhy vyžadovanej pri ich kreslení, presune. Spoločný rodič zabezpečuje homogénnosť všetkých dediacich tried, kôli potrebe uloženia všetkých objektov do jediného spojového zoznamu. Každá inštancia jednej z týchto piatich dediacich tried obsahuje základné informácie o počte vstupných a výstupných pinov, spojoch, umiestnení v schéme, type, názve, prototype objektu, vie sa vykresliť do pohľadu.

4.1.3. Pomocné triedy

V následujúcej časti textu sú popísané pre aplikáciu dôležité pomocné triedy a ich význam.

4.1.3.1. Trieda `CElementContainer`

Táto trieda je jednou zo základných zložiek aplikácie. Realizuje vytváranie prototypov grafických objektov načítaním informácií zo súboru formátu XML. Pre načítanie XML využíva open-source c++ XML parser, TinyXml. Z takto načítaných údajov vytvára inštancie tried dediacich z triedy `CCommonObject`, vytvára teda prototypy a zároveň ich ukladá do vlastných premenných typu asociatívna mapa. Inštancia tejto triedy sa nachádza v samotnej aplikácii (trieda CWinApp) ale aj v každom otvorenom dokumente aplikácie. Aplikáciou je využívaná pre uloženie preddefinovaných prototypov logických hradiel, blokov a taktiež obsahuje aj dva prototypy pre vstupné a výstupné značky. Každý z dokumentov využíva objekt tejto triedy pre uloženie prototypov objektov špecifických projektu. Pri načítaní grafického zobrazenia objektov sú rozpoznávané grafické elementy čiara (line), oblúk (arc), elipsa (ellipse), bézierové krivky (bezier), text (text). Z týchto grafických elementov je vytvorená grafická cesta, teda jeden celok, ktorý je aplikácia schopná vykresliť do pohľadu.

Formát špecifikácie prototypu elementu prijímaný aplikáciou:

```
<GELEMENT>
  <TYPE>Gate</TYPE>
  <NAME>AND</NAME>
```

Táto špecifikácia predstavuje grafický element typu Gate, teda hradlo s menom AND. Nasledujú informácie týkajúce sa spôsobu zobrazenia. Ten je rozdelený na špecifikáciu 'tela' objektu (body) a výstupných (outputs) resp. vstupných (inputs) častí objektu.

```
  <DRAWING>
    <BODY nRecords="5">
      <LINE x1="20" y1="0" x2="60" y2="0">line</LINE>
      <BEZIER x1="60" y1="0" x2="100" y2="0" x3="100" y3="40" x4="60"
y4="40">bezier</BEZIER>
      <LINE x1="60" y1="40" x2="20" y2="40">line</LINE>
      <LINE x1="20" y1="40" x2="20" y2="0">line</LINE>
      <CLOSEFIGURE>closefigure</CLOSEFIGURE>
    </BODY>
    <OUTPUTS nOut="1" nRecords="2">
      <LINE x1="90" y1="20" x2="110" y2="20">line</LINE>
```

```
      <CLOSEFIGURE>closefigure</CLOSEFIGURE>
    </OUTPUTS>
    <INPUTS nIn="2" nRecords="4">
      <LINE x1="0" y1="10" x2="12" y2="10">line</LINE>
      <CLOSEFIGURE>closefigure</CLOSEFIGURE>
      <LINE x1="0" y1="30" x2="20" y2="30">line</LINE>
      <CLOSEFIGURE>closefigure</CLOSEFIGURE>
    </INPUTS>
  </DRAWING>
```

Kvôli rôznorodosti možných grafických reprezentácií objektu je potrebné explicitne definovať miesto vstupného a výstupného pinu, teda miesto napojenia objektu na spoj. Nejde už teda o grafické informácie, ale o informácie súvisiace s funkcionalitou objektu. Zároveň je potrebné špecifikovať, či ide o vstup (resp. výstup) invertovaný, alebo nie.

```
  <COBJINFO>
    <OUTPUTRECT nRecords="1">
      <OUTPUT x1="110" y1="20" inv="0">0</OUTPUT>
    </OUTPUTRECT>
    <INPUTRECT nRecords="2">
      <INPUT x1="0" y1="10" inv="0">0</INPUT>
      <INPUT x1="0" y1="30" inv="0">1</INPUT>
    </INPUTRECT>
  </COBJINFO>

</GELEMENT>
```

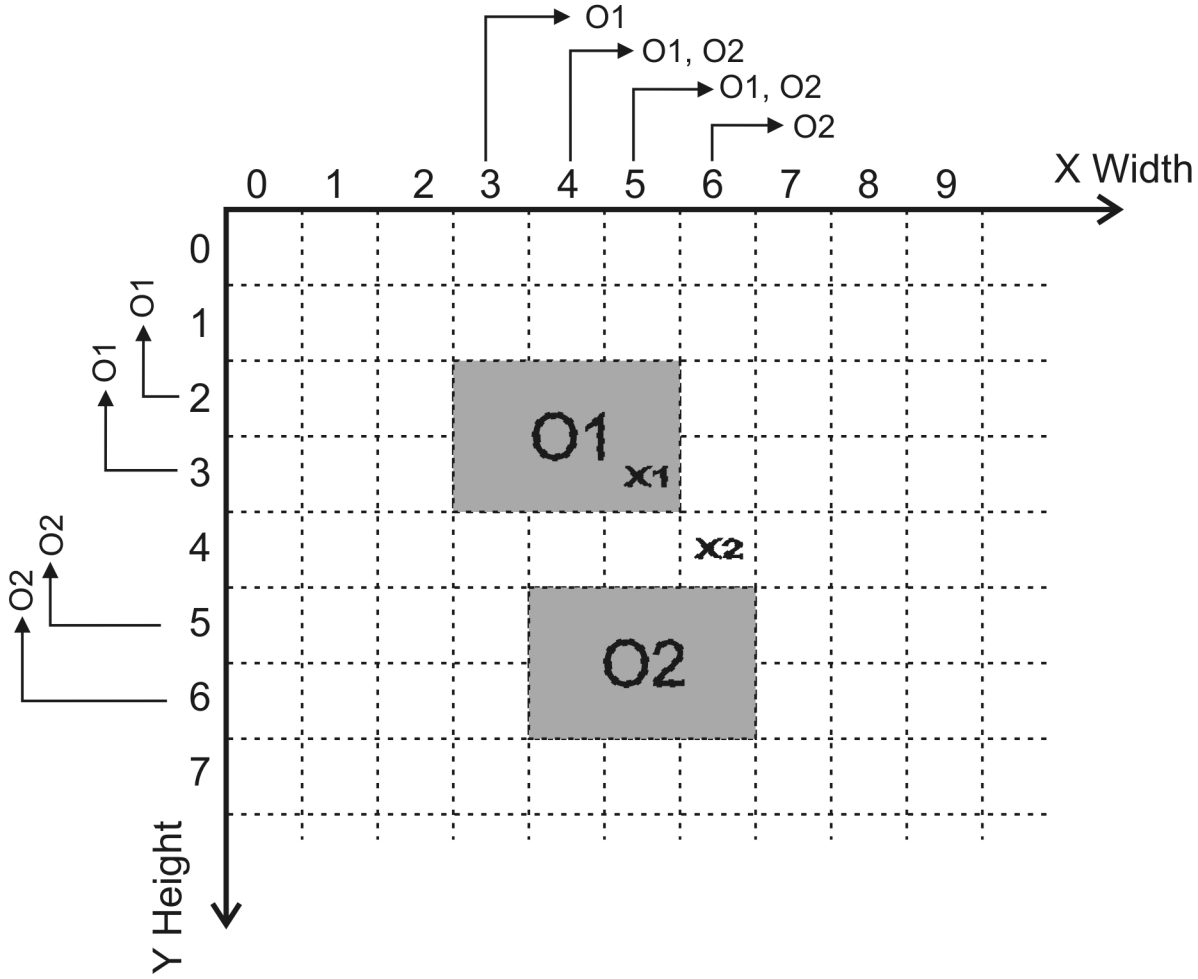
4.1.3.2. Trieda `CMouseTracker`

Inštancia tejto triedy sa nachádza v každom otvorenom dokumente a obsahuje funkcie pre obsluhu podnetov vytváraných myšou pri kreslení schémy. Po kliknutí ľavého tlačidla myši na kresliacu plochu, napríklad ako podnetu pre vloženie objektu, alebo prípadného podnetu na premiestnenie existujúceho objektu sú Windows správy (Windows Messages) smerované cez objekt tejto triedy. Ten odchytáva

správy špecifické pre myš a spracúva ich. Základný princíp takéhoto spôsobu obsluhy správ myši bol prevzatý z [3]. Medzi úlohy funkcií tejto triedy patrí napríklad správa kurzorov myši pri rôznych režimoch kreslenia, zobrazenie konceptu náčrtu objektu pri jeho presúvaní a taktiež hit-test na vstupné, resp. výstupné piny objektu.

4.1.3.3. Trieda `CSpacialHashing`

Podstatou významu tejto triedy je registrácia objektov v schéme za účelom zvýšenia efektivity označovania objektov.



Orázok 6. – Princíp registrovania objektov triedou CSpacialHashing

Plocha využitá na kreslenie je rozdelená na horizontálnu a vertikálnu časť (X Width je šírka tejto plochy a Y Height jej výška) predstavovanú dvoma poľami pevnej veľkosti, horizontálnym a vertikálnym, vytvorenými pri spustení aplikácie podľa rozmerov

kresliacej plochy, typu `CHashItem`. Ide o triedu vytvárajúcu jednoduchý spojový zoznam odkazov na objekty registrované na konkrétnom indexe poľa. Pri umiestnení objektu (O1 a O2) do schémy je jeho poloha registrovaná touto triedou tak, že odkaz na vkladaný objekt je pridaný do spojového zoznamu všetkých indexov ktoré prekrýva svojím zobrazením. Takýto spôsob registrácie objektov je veľkým časovým zvýhodnením práce s objektami pri hit-teste objektu. Ten je pozitívny v prípade, že v bode kresliacej plochy, kde uživateľ klikne myšou, je registrovaný jeden objekt v horizontálnom aj vertikálnom smere (X1). Je negatívny v prípade, že v mieste kliknutia je objek regitrovaný len v horizontálnom, prípadne len vo vertikálnom (X2) smere, alebo nieje registrovaný vôbec. Bez využitia tejto triedy by bolo hit-test nutné prevádzať s asymptotickou zložitosťou minimálne O(n), predstavujúcou sekvenčené prechádzanie spojového zoznamu, zvýšenou o následný hit-testom každého objektu v schéme pomocou funkcií GDI+. S využitím tejto triedy ide o konštantný čas hit-testu.

4.1.3.4. Triedy `CXmlCreator` a `CPSCreator`

Úlohou týchto tried je vytvorenie výstupu z nakreslenej schémy. V prípade triedy `CXmlCreator` táto dokáže vytvárať súbory typu XML. Využíva sa teda pri vytvorení textového popisu projektu, schém obsiahnutých v projekte a objektov obsiahnutých v schémach. Štruktúra takto vytvorených súborov pre schémy a projekty je podobná.

Formát špecifikácie projektu:

```
<CONTENT>
      <ELEM type="1">
            <DISPLAY>Scheme</DISPLAY>
            <ORIGIN inProject="1">Scheme</ORIGIN>
      </ELEM>
      <ELEM type="1">
            <DISPLAY>Scheme2</DISPLAY>
            <ORIGIN inProject="1">Scheme2</ORIGIN>
      </ELEM>
</CONTENT>
```

Formát špecifikácie schémy:

```
<CONTENT>
<ELEM type="3">
      <DISPLAY>Gate1</DISPLAY>
      <ORIGIN inProject="0">OR</ORIGIN>
      <NAME>Gate1</NAME>
      <POSITION tlx1="140" tly1="50"></POSITION>
      <CONNECTIONS>
            <INCONNECTIONS nCount="2">
                  <IN Name="InTag0">0</IN>
                  <IN Name="InTag1">1</IN>
            </INCONNECTIONS>
            <OUTCONNECTIONS nCount="1">
                  <OUT Name="OutTag0">0</OUT>
            </OUTCONNECTIONS>
      </CONNECTIONS>
</ELEM>
...
</CONTENT>
```

Formát projektu aplikácie predstavuje zoznam schém. Schéma je predstavovaná zoznamom objektov v nej obsiahnutých, spolu so špecifikáciou objektu pozostávajúcou z typu (atribút type), mena unikátneho vrámci schémy (Name), názvu prototypu (Origin), polohy ľavého horného rohu v schéme, a taktiež názvov objektov pripojených na ich vstupné resp. výstupné piny spolu s poradovým číslom udávajúcim miesto ich pripojenia. Formát súborov reprezentujúcich funkčné bloky pridáva k formátu schémy, teda vlastnému zloženiu bloku ešte popis prototypu bloku.

4.2. Štruktúra aplikácie

Pomocné triedy predstavujú časť implementácie aplikácie vykonávajúcu operácie na pozadí, uloženie dočasne vytvorených objektov, sú základom funkčnosti aplikácie. Druhou časťou je vlastné zobrazenie, teda konkrétne grafická reprezentácia možností aplikácie uživateľovi a grafických objektov uživateľom nakreslených. Takáto architektúra, všeobecne známa ako MVC (Model View Controller) zprehľadňuje

výsledný zdrojový kód a zvyšuje predpoklad správnel lokalizácie prípadnej chyby v ňom. V prípade MFC (Microsoft Foundation Classes) je tento spôsob nazývaný Document/View architecture. K časti Document (Model) patria pomocné triedy, trieda `CINKSDoc` a trieda `CTreeBar`, ktoré obsahujú všetky aplikáciou zobrazované dáta. View aplikácie je tvorený triedami `CSideBar`, predstavujúcou zobrazenie grafických objektov, ktoré môže uživateľ kresliť, triedou `CTreeBar`, ktorá v sebe zahŕňa vlastný stromový model hierarchického návrhu schémy ktorý je kópiou obsiahnutých grafických objektov a samotnou MFC triedou `CINKSView`, tvoriacu kresliacu plochu, na ktorej sú grafické objekty kreslené.

5. Záver

Výsledný produkt tohto projektu predstavuje aplikáciu, pomocou ktorej je možné kresliť schémy logických obvodov s použitím preddefinovaných grafických objektov reprezentujúcih logické hradlá, bloky, značky, Podporuje možnosť vytvorenia bloku z nakreslenej schémy, s možnosťou tento blok upravovať, znamená to, že podporuje hierarchiu návrhu. Schému je možné uložiť do súboru, spätne načítať aplikáciou a znovu upravovať. Výsledné úspechy je možné formulovať takto:

- Kreslenie schémy použitím preddefinovaných symbolov
- Uloženie a znovunačítanie schémy
- Podpora hierarchie schémy, vytváranie funkčných blokov
- Postscriptový výstup nakreslenej schémy

Výsledkom je teda aplikácia použiteľná pre tvorbu schémy. Slabou stránkou aplikácie je práca so spojmi, predstavujúcimi komplexnú oblasť (automatické spojovanie, úprava jednotlivých častí spoja, vymazanie časti spoja ai.) vyžadujúcu zložitú obsluhu potrebnú pri prípadnom požiadavku čo najjednoduchšej práce s nimi. Nakoľko bol tento projekt jeden z mojích prvých projektov tak značného rozsahu, myslím si, že zadanie sa mi splniť podarilo. Verím, že vďaka mojej snahe o prehľadnosť a logické usporiadanie zdrojového kódu nebudú nastávať problémy pri jeho budúcom rozšírení prípadnou inou osobou.

5.1. Budúca práca

Je mnoho oblastí, v ktorých môže byť aplikácia vylepšená a rozšírená. Medzi hlavné patrí efektívnejšia práca so spojmi, správa pamäti, rozšírenie možnosti výstupov aplikácie, alebo aj rozšírenie funkcionality aplikácie.

5.1.1. Práca so spojmi

Vylepšenie tejto oblasti aplikácie predstavuje veľmi zložitý problém vyžadujúci veľmi prepracovaný a rozsiahly spôsob automatického spojovania pinov, ako napríklad pri tvorbe spoja, ale aj pri premiestňovaní objektu. Základy pre takéto rozšírenie sú

položené, existuje totiž aplikáciou vytvorené akési 'bludisko' tvorené objekami v schéme, v ktorom je potrebné nájsť v najjednoduchšom prípade spojenie dvoch bodov najkratšou cestou, avšak je samozrejmé, že tento problém siaha do oveľa väčšej hĺbky.

5.1.2. Správa pamäti

Aplikácia napriek nemalej snahe stále obsahuje chyby vzniknuté nesprávnou dealokáciou pamäti. Je pochopiteľné, že táto chyba je dosť podstatným problémom, a bolo by veľmi vhodné odstrániť ju.

5.1.3. Rozšírenie možností výstupov

Táto oblasť budúceho zamerania nieje nevyhnutná, avšak z dôvodu priateľského prostredia aplikácie potrebná. Vhodným pridaním by mohol byť výstup do multiplatformového formátu PDF, alebo rastrový výstup.

Obsah priloženého CD

Na priloženom CD sa nachádzajú adresáre s nasledujúcim obsahom:

- Koreňový adresár - obsahuje súbor readme.txt popisujúci mapu CD.
- \Documentation - obsahuje tuto bakalársku prácu vo formáte PDF, uživateľskú príručku aplikácie.
- \bin - obsahuje spustiteľný súbor aplikácie.
- \bin\additional - obsahuje podadresáre blocks a gates kde sa nachádzajú preddefinované prototypy funkčných blokov a hradiel.
- \source - obsahuje zdrojové súbory.

Použitá literatúra a zdroje

[1] Rubin, Steven M. "Computer Aids for VLSI Design", Published on the World Wide Web in 1997, http://www.rulabinsky.com/cavd/, kapitola 3

[2] Mark D. Birnbaum "Essential Electronic Design Automation (EDA)", Prentice Hall, October 01 2003, kapitola 4

[3] Sjaak Priester "Drawing Curved Objects", September 08, 2005 , http://www.codeguru.com

6. Príloha

6.1. Popis API

Táto časť textu krátko popisuje funkcie jednotlivých tried pre prípad úpravy zdrojového kódu inými uživateľmi. Neobsahuje popis funkcií, ktoré sú súčasťou MFC a zostali nepozmenené, ani funkcie, ktorých význam je možné veľmi jednoduché zistiť.

6.1.1. Trieda `CCommonObject`:

`virtual void ResetGPath();`
Funkcia inicializuje grafickú cestu (GDI+) daného objektu.

`virtual void TransformCObject(Matrix* pMat);`
Funkcia transformuje daný objekt pomocou transformačnej matice pMat.

`virtual void OutScreen(Graphics &g);`
Funkcia pre výstup daného objektu do pohľadu aplikácie, funguje nazáklade vloženia grafickej cesty do `Graphics` objektu g. Volaná triedou `CINKSView`.

`virtual CCommonObject* ConnectHitTest(CPoint point, int &nNumber);`
Funkcia predstavuje hit-test metódu pre pripojenie kresleného spoja na vstupný, alebo výstupný pin objektu. Je volaná triedou `CMouseTracker`, nastavuje hodnotu `nNumber` na +poradové číslo vstupu, -poradové číslo výstupu, alebo 0 v prípade neúspechu hit-testu.

`virtual CCommonObject* Clone(CPoint TopLeftOrigin, CString strNewName, int nRotation);`
Funkcia vracia klon objektu, spolu s nastavenou novou pozíciou na základe bodu `TopLeftOrigin`, s novým menom `strNewName` a rotáciou objektu `nRotation` v stupňoch. Je využívaná pre kopírovanie prototypov pri kreslení schémy.

`virtual CString GetObjectName();`
Funkcia vráti meno daného objektu (unikátne vrámci schémy)

`virtual Rect* GetBounds();`
Funkcia vracia ohraničujúci obdĺžnik grafického objektu, využívaného pri jeho registrácii triedou `CSpacialHashing`.

```
virtual void SetConnection(bool bIn, int nPos, CCommonObject*
pConnectTo);
```
Funkcia nastaví spojenie daného objektu s objektom `pConnectTo`, hodnota `bIn` určuje, či sa jedná o nastavenie niektorého zo vstupných pinov, hodnota `nPos` určuje pozíciu tohto pinu.

```
virtual void DisconnectCObject(CCommonObject *pObject);
```
Funkcia odpojí od daného objektu objekt `pObject`.

```
virtual void Disconnect();
```
Funkcia odpojí daný objekt od všetkých ostatných objektov.

```
int GetType();
```
Funkcia vracia číselný typ daného objektu.

```
virtual void SetRotation(int nRotation);
```
Funkcia nastavuje rotáciu daného objektu o `nRotation` stupňov.

Triedy dediace z tejto triedy (konkrétne ide o `CGate`, `CBlock`, `CTag`, `CWire`, `CInv`) využívajú predefinované definície týchto funkcií, každá podľa vlastnej potreby.

6.1.2. Trieda CINKSApp

```
void CreateNewElement(int nType);
```
Obslužná funkcia sprostredkúvajúca vytvorenie nového projektu, schémy, alebo bloku.

```
void BringToTop(CDocument *pDoc);
```
Funkcia pre aktiváciu dokumentu aplikácie `pDoc`.

```
void CreateNewProject();
```
Obslužná funkcia vytvárajúca nový projekt vytvorením potrebných adresárov a projektového XML súboru.

```
void CreateNewScheme(CTreeItem *pParent);
```
Obslužná funkcia vytvárajúca novú schému, objekt `pParent` špecifikuje rodiča tejto schémy pre zobrazenie v stromovej štruktúre.

```
void CreateNewBlock();
```
Obslužná funkcia vytvárajúca nový blok z označenej schémy, spolu s vytvorením XML prototypu bloku.

```
void CloseAllDocuments(BOOL bEndSession);
```

Funkcia zatvárajúca všetky otvorené dokumenty. Na základe hodnoty `bEndSession`. Pre hodnotu `false` ich skryje, pre hodnotu `true` ich zatvorí a zničí ich obsah.

```
CDocument* OpenDocumentFile(LPCTSTR lpszFileName);
```

Funkcia otvárajúca projekt, ktorého cesta je špecifikovaná `lpszFileName` a vráti ukazateľ na novovytvorený dokument. Pohľad asociovaný k tomuto dokumentu je skrytý.

```
CDocument* CreateDocView(CString strFilename, CString strTitle, BOOL bViewVisible);
```

Funkcia vytvárajúca nový dokument a jemu priradený pohľad, nastaví cestu k súboru ktorý reprezentuje na `strFilename` a názov dokumentu na `strTitle`. Podľa hodnoty `bViewVisible` bude tento pohľad viditeľný.

6.1.3. Trieda CINKSDoc

```
BOOL OnOpenFile(CString strFile, int nType, bool bInProject = false);
```

Funkcia otvárajúca schému, alebo funkčný blok určený súborom s cestou `strFile` a typom `nType`, spolu so špecifikáciou, či ide o projektový súbor, alebo nie. Vracia TRUE v prípade úspechu, FALSE v prípade neúspechu.

```
CCommonObject* CreateNewCommonObject(CPoint point, CString &strOrigin, int nSource);
```

Funkcia vytvárajúca nový grafický objekt klonovaním prototypu s názvom `strOrigin`, ktorý je umiestnený na pozíciu point. Hodnota `nSource` špecifikuje pôvod prototypu (projektový, alebo aplikačný). V prípade nájdenia prototypu a úspešného klonovania funkcia vracia ukazateľ na nový objekt, v inom prípade vracia NULL.

```
bool AddNewCommonObject(CPoint point);
```

Obslužná funkcia pre pregistráciu novovytvoreného grafického objektu v triede `CSpacialHashing` a `CTreeBar`, určenej ľavým horným rohom objektu `point`. Vracia hodnotu `true` v prípade vytvorenia a registrácie objektu, alebo hodnotu `false` pri chybe, ktorá sa mohla vyskytnúť pri klonovaní prototypu, registrácii objektu, alebo pri neúspešnom výsledku testovania dostačujúceho voľného miesta v schéme pre umiestnenie objektu.

```
bool AddNewCommonObjectWire(CPoint point);
```

Funkcia vytvárajúca nový objekt typu `CWire`, teda spoj, spolu s jeho registráciou v triede `CSpacialHashing` a `CTreeBar`.

```
void DeleteSelection();
```

Obslužná funkcia pre vymazanie uživateľom označených objektov zo spojového zoznamu všetkých objektov. Zároveň pri dealokácii pamäti je objekt deregistrovaný triedou `CSpacialHashing` a taktiež `CTreeBar`.

```
void UpdateObjecInfo();
```

Funkcia slúžiaca na zobrazenie informácií o označenom objekte.

```
void TryMovedConnection();
```

Funkcia volaná pri premiestnení objektu, ktorej úlohou je zistiť nové napojenia grafických objektov, ktoré pri premiestnení mohli vzniknúť.

```
void ConnectElements(CList<CString*, CString*> *inList,
CList<CString*, CString*> *outList);
```

Funkcia vytvárajúca prepojenia medzi objektami v schéme pri otváraní existujúcej schémy, alebo funkčného bloku, kde spojový zoznam `inList` resp. `outList` predstavuje názvy objektov deklarovaných v textovom popise ako pripojených na vstup resp. výstup nejakého objektu.

```
CCommonObject* GetCommonObject(CString strName);
```

Funkcia vracia odkaz na objekt s názvom `strName`. V prípade, že sa v spojovom zozname danej schémy, alebo funkčného bloky objekt s týmto názvom nenachádza, vracia hodnotu NULL.

```
BOOL OnSaveDocument(LPCTSTR lpszPathName);
```

Funkcia prevádzajúca uloženie vnútorných informácií do súboru, vytvára teda XML súbor s cestou `lpszPathName`, do ktorého je zapísaný obsah schémy.

```
bool SOListContains(CCommonObject *pObj);
```

Funkcia vracia hodnotu `true` v prípade, že spojový zoznam objektov označených uživateľom obsahuje objekt `pObj`, hodnotu `false` v opačnom prípade.

6.1.4. Trieda CElementContainer

```
bool OpenProject(CString strProject);
```

Funkcia kontrolujúca existenciu a validitu otváraného súboru špecifikujúceho projekt. Na základe tejto hodnoty vracia `true` v prípade úspechu, `false` v opačnom prípade.

```
bool OpenProjectSpecific(CString strElementName, CString
```

`strProjectPath);`

Funcia pre načítanie prototypu s názvom `strElementName` definovaným v projektovom adresári `strProjectPath`, prípadne niektorom z adresárov v ňom obsiahnutých. Hodnota, ktorú vracia, je na základe kontroly existencie prototypu v danom adresári. Načítaný prototyp je uložený do mapy prototypov s kľúčom `strElementName`.

`void OpenProjectSpecific(CString strProjectPath, int nWhich);`

Funkcia s významom podobným predošlej, realizujúca načítanie prototypov v adresári projektu `strProjectPath`. Hodnota `nWhich` špecifikuje, či sa jedná o načítanie všetkých prototypov (hodnota 0), všetkých prototypov hradiel (hodnota 1), alebo všetkých prototypov blokov (hodnota 2).

`bool OpenProjectSpecific(CString strFilename);`

Funkcia realizujúca načítanie protypu konkrétneho súboru špecifikovaného cestou `strFilename`.

`void OpenAppSpecific(CString strAppPath, int nWhich);`

Funkcia rovnakého významu ako `OpenProjectSpecific`, volaná aplikáciou pre načítanie prototypov špecifických aplikácii.

`bool OpenAppSpecific(CString strElemName);`

Funkcia rovnakého významu ako `OpenProjectSpecific`, volaná aplikáciou pre načítanie prototypu špecifického aplikácii.

`bool ContainsElement(CString strElementName);`

Pomocná funkcia, vracia true v prípade, že mapa prototypov obsahuje prototyp s názvom `strElementName`, false v opačnom prípade.

`bool CanAppContainElement(CString strElementName);`
`bool CanProContainElement(CString strElementName);`

Pomocné funkcie pre zistenie existencie definície prototypu s názvom `strElementName` v projektovom adresári, resp. v adresári aplikácie.

`bool CanContainElement(CString strElementName);`

Funkcia predstavuje spojenie funkcií `bool CanAppContainElement(CString strElementName)` a `bool CanProContainElement(CString strElementName)`

`CString AssociateFile(CString strElementName, CString strPath);`

Funkcia vyhľadá názov súboru definujúceho prototyp s názvom `strElementName` v adresári `strPath`. Vráti jeho cestu v prípade nájdenia, v opačnom prípade prázdny

string.

`CCommonObject *GetCommonObject(CString strElement);`

Funkcia vracia odkaz na prototyp objektu s názvom `strElement` uloženého v mape prototypov, vracia NULL v prípade jeho neexistencie.

6.1.5. Trieda CMouseTracker

`int Track(CDC *pDC, UINT nFlags, CPoint point);`

Funkcia volaná pri každom kliknutí tlačidla myši predstavujúceho následujúcu prácu s grafickým objektom (jeho vytváranie, premiestnenie,...), ktorá odchytáva Windows Messages patriace aplikácii a spracúva správy vytvárané myšou. Na základe typu tejto správy volá obslužné funkcie. Funkcia prijíma ukazateľ na `CDC`, ktorý vpodstate predstavuje kresliacu plochu, počiatočné flagy stavu myši `nFlags` a bod kliknutia myšou `point`. Funkcia vracia číselný výsledok procesu sledovania myši signalizujúci uspešné ukončenie procesu, presun objektu, vytvorenie nového objektu, označenie viacerých objektov, úpravu spoja, zrušenie procesu užívateľom, alebo neúspech procesu na základe nejakej vzniknutej chyby.

`virtual int OnBeginTrack(UINT nFlags, CPoint point);`

Funkcia inicializujúca proces sledovania myši, volaná funkciou `int Track(CDC *pDC, UINT nFlags, CPoint point)` na jej začiatku. Inicializácia prebieha na základe zvoleného módu kreslenia aplikácie, teda či sa napríklad jedná o vytváranie spoja, umiestnenie nového grafického elementu ai.

`virtual int OnEndTrack(int trackResult);`

Funkcia ukončujúca proces sledovania myši, volaná funkciou `int Track(CDC *pDC, UINT nFlags, CPoint point)` na jej konci.

`void Draw(CDC *pDC, Graphics &g);`

Funkcia využívaná pre vykreslenie náčrtu grafického objektu pri jeho presúvaní, prípadne náčrtu značky pri jej kreslení.

`void SelectMultiple(CPoint point);`

Funkcia kresliaca náčrt obdĺžniku pri označovaní viacerých grafických objektov schémy.

`bool AdaptWire(CPoint from, CPoint to, AnchorList &addTo);`

Funkcia pre jednoduché spojenie dvoch bodov, bodu from a bodu to vytváraného úsečkami na seba kolmými so snaho nekrížiť žiaden z objektov schémy. Body vytvárané týmto procesom sú pridávané do spojového zoznamu bodov spoja `addTo`.

6.1.6. Trieda CSpacialHashing

`void RegisterCObject(CCommonObject *pObject, int l, int t, int r, int b);`

Funkcia prevádza registráciu grafického objektu pObject na oblasti ohraničenej bodmi `l`, `t`, `r`, `b` predstavujúcimi ľavý, horný, pravý a spodný bod oblasti.

`void RegisterWire(CCommonObject *pObject, Point *points, int nPointCount);`

Funkcia prevádza registráciu spoja `pObject` na základe jeho bodov `points`.

`void DeRegisterCObject(CCommonObject *pObject, int l, int t, int r, int b);`

Funkcia s významom opačným ako funkcia pre registráciu grafického objektu.

`void DeRegisterWire(CCommonObject *pObject, Point *points, int nPointCount);`

Funkcia s významom opačným ako funkcia pre registráciu spoja.

`void MoveCObject(CCommonObject *pObject, CPoint oldTL);`

Funkcia prevádzajúca deregistráciu oblasti uvoľnenej pri presune grafického objektu `pObject` a následnú registráciu novoobsadených pozícií. Hodnota `oldTL` špecifikuje pôvodnú pozíciu ľavého horného rohu grafického objektu.

`void MoveWire(CCommonObject *pObject, Point *oldPoints, int nOldPointCount, Point *newPoints, int nNewPointCount);`

Funkcia podobného významu ako `void MoveCObject(CCommonObject *pObject, CPoint oldTL)`, avšak pre objekty typu spoj.

`CCommonObject *HashTest(CPoint point);`

Funkcia realizujúca metódu hit-test pre bod `point`. Vracia konkrétny ukazateľ na grafický objekt, alebo hodnotu NULL v prípade neúspešnosti hit-testu.

`int GetHLine(int Vpos, int start, int end);`

Pomocná funkcia pre prácu so spojmi pri automatickom prepojovaní dvoch bodov. Hodnota `Vpos` určuje vertikálnu polohu umiestňovaného spoja a hodnoty start resp. end určujú počiatočnú resp. koncovú polohu bodu vytváraného spoja. Funkcia vracia maximálnu možnú dĺžku vytváraného horizontálneho spoja z bodu `start` bez kríženia sa s iným objektom v schéme. Vracia hodnotu `end` v prípade možnosti neprerušeného spojenia bodov `start` a `end`.

`int GetVLine(int Hpos, int start, int end);`

Funkcia pracujúca na princípe podobnom funkcii `int GetHLine(int Vpos, int`

`start, int end)`, hodnota `Hpos` však určuje horizontálnu polohu umiestnenia spoja a funkcia vracia maximálnu možnú dĺžku vytváraného vertikálneho spoja z bodu `start` bez kríženia sa s iným objektom v schéme.

```
bool HashAreaTest(CPoint point, int width, int height);
```

Funkcia slúžiaca pre zistenie možnosti vloženia objektu s ľavým horným rohom `point` grafického objektu, šírkou `width` a výškou `height`. Vracia `true` v prípade, že plocha parametrami definovaná nieje obsadená žiadnym iným objektom, `false` v prípade, že táto plocha obsadená nejakým objektom je.